TAMPEREEN
AMMATTIKORKEAKOULU

KONFIGURAATIOHALLINTA HAJAUTETUSSA JÄRJESTELMÄSSÄ

Jani Timonen

Opinnäytetyö
Joulukuu 2017
Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutus
Ohjelmistotuotanto

TIMONEN, JANI
Konfiguraatiohallinta hajautetussa järjestelmässä

Opinnäytetyö 32 sivua, joista liitteitä 2 sivua
Joulukuu 2017

Tämän opinnäytetyön tavoitteena oli kehittää toimeksiantajan toimintamallia nykyisten järjestelmien konfiguraatiohallinnassa. Tarkoituksena oli rakentaa toimiva ohjelmistokomponentti, jonka avulla voidaan hallita konfiguraatioita, sekä jakaa niitä muihin järjestelmiin.

Konfiguraatiot ovat oleellinen osa suuremman järjestelmän toimintaa. Niiden hallintaan on hyvä kehittää toimiva toimintamalli, jotta voidaan välttyä turhalta työtaakalta. Parhaimmillaan sujuva konfiguraatiohallinta mahdollistaa ohjelmiston toiminnan muuttamisen nopeasti ja tehokkaasti ilman koodiin tehtäviä muutoksia. Varsinkin suuren kokoluokan hajautetussa järjestelmässä voi esiintyä ongelmia joiden ratkaiseminen on tärkeää. Työntekijä voi joutua käyttämään paljon työaikaa vain pienen muutoksen takia.

Toimivan konfiguraatioiden hallintaan erikoistuvan järjestelmän rakentamiseen on tarjolla lukuisia teknologioita. Näistä teknologioista voidaan valita tehokkaita ja tarkoitukseen sopivia. Erilaiset suorituskykytestit tarjoavat tietoa siitä, onko kyseinen teknologia tarkoituksiin sopiva. On hyvä muistaa, että erilaisiin tarkoituksiin sopivat erilaiset teknologiat eikä voida sanoa, että yksi on ylitse muiden. Eri teknologioiden avulla voidaan rakentaa toimiva järjestelmä, jonka avulla konfiguraatiohallinta on tehokasta ja helppoa.

Toimiva ohjelmistokomponentti saatiin rakennettua. Sen hienosäätäminen ja jatkokehittäminen tehokkaaksi on kuitenkin jatkuva prosessi. Myös uudet käyttötarkoitukset tulevat vaatimaan uusia toiminnallisuuksia ja vanhojen jalostamista.

Asiasanat: konfiguraatio, hajautettu järjestelmä, palvelukeskeinen arkkitehtuuri

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Software Production

TIMONEN, JANI:
Configuration Management in Distributed Systems

Bachelor's thesis 32 pages, appendices 2 pages
December 2017

The object of this thesis was to improve procedures with configuration management in the company. The purpose was to design and implement a working system which could help making configuration management easier.

Configurations are important part of a large-scale software system. In the best situation they can make the software more flexible without a need of code changes. Configuration management is important. Without proper procedures changing configurations can require unnecessary work from an employee. This can be a big problem factor when working with large-scale systems.

There are many technologies available to build a working system to manage configurations. Most of these have a wide variety of benchmark tests. By using these tests the developers can select technology that is best suited for the purpose. A developer should remember that each technology has been designed for certain purposes, and for this reason it is not possible to choose one that is best for all tasks. By utilizing the strengths of each technology, a working and efficient system can be built to manage and distribute configurations.

In this project a working software component was built successfully. Fine tuning and developing the system to be more efficient is a longer process. Especially new use cases require creation of new functionalities and modification of existing ones.

Key words: configuration, distributed systems, service oriented architecture

SISÄLLYS

LYHENTEET JA TERMIT

Hajautettu järjestelmä	ohjelmisto, jonka osat sijaitsevat toisistaan erotettuina esimerkiksi erillisillä palvelimilla
Instanssi	ohjelmasta käynnissä oleva yksittäinen prosessi
Java	olio-ohjelmointikieli
Jar-paketti	Javan paketointimuoto, joka voidaan ajaa tai liittää toiseen Java-projektiin
Konfiguraatio	määritelmä, joka vaikuttaa ohjelman toimintaan ajon aikana.
SOA	palvelukeskeinen arkkitehtuuri (Service Oriented Architecture)

1 JOHDANTO

Konfiguraatiot ovat tärkeä osa laajemman ohjelmiston kehittämistä. Niiden avulla voidaan muokata helposti ohjelmiston toimintaa ilman koodiin tehtäviä muutoksia. Konfiguraatiot voivat yksinkertaisimmillaan olla tekstitiedostossa olevia arvoja, joita muokkaamalla ohjelma saadaan toimimaan eri tavalla.

Tämän opinnäytetyön toimeksiantaja kehittää ja ylläpitää laajoja hajautettuja järjestelmiä. Suuren kokoluokan vuoksi nykyinen toimintamalli konfiguraatioiden hallinnassa ei ole tarpeeksi tehokas. Heräsi ajatus, että asialle voidaan tehdä ja onkin tehtävä jotain. Päätettiin ideoida uusi toimintamalli tehokkaaseen ja helposti skaalautuvaan konfiguraatiohallintaan. Tämän konfiguraatiohallinan keskipisteenä toimisi järjestelmä, joka tarjoaa toiminnallisuuksia tehokkaaseen konfiguraatiohallintaan. Tätä järjestelmää hyödyntäen voidaan hallinnoida hajautettujen järjestelmien konfiguraatiota tehokkaasti ja keskitettynä.

Opinnäytetyön tavoitteena on löytää nykyisen toimintamallin ongelmakohtia ja kehittää toimintamallia tehokkaampaan ja paremmin skaalautuvaan suuntaan. Tarkoituksena on luoda toimiva ohjelmistokomponentti, joka edesauttaa uuden toimintamallin toteutumista. Ohjelmistokomponentin on tarkoitus olla helposti liitettävissä nykyiseen hajautettuun järjestelmään.

Opinnäytetyössä käytetään esimerkkinä verkkokauppasovellusta, joka kuvastaa rakenteellisesti toimeksiantajan järjestelmiä. Varsinaista toimeksiantajan järjestelmää ei voida luottamuksellisuussyistä käyttää esimerkkinä.

2 TEKNOLOGIAT

Tässä luvussa käydään läpi teknologioita, joita kehitetty ohjelmistokomponentti hyödyntää. Luvussa esitellään, mistä teknologiassa on kyse, ja erityisesti käydään läpi, miksi juuri kyseinen teknologia on valittu mukaan tähän opinnäytetyöhön. Keskeisenä kriteerinä oli löytää teknologioita, jotka tekevät yhden asian hyvin. Näin saadaan järjestelmä, jossa ei ole ylimääräisiä toimintoja, jolloin ongelmatilanteet voidaan minimoida mahdollisimman vähäisiksi.

2.1 Konfiguraatioiden tallennus

Ensimmäinen valittava teknologia oli varsinaisten konfiguraatioiden tallennuspaikka. Melko yksinkertainen vaihtoehto olisi ollut perinteinen tietokanta, mutta haluttiin yrityksen käytäntöjen mukaan tutkia mahdollisia muita vaihtoehtoja. Oli melko selvää, että tarvittavan teknologian tulisi kyetä tallentamaan nykyisiä konfiguraatioita. Nykyiset konfiguraatiot ovat yksinkertaisia avainarvopareja. Ohjelmiston konfiguraatioilla tarkoitetaan helposti muutettavia arvoja, joiden mukaan ohjelmiston tai järjestelmän toiminta muuttuu. Kuten moneen muuhunkin käyttötarkoitukseen, on tämänkin järjestelmän tuottamiseen vaadittujen teknologioiden tarjonta valtava. Valintaan vaikuttivat vahvasti yrityksen vaatimukset sekä kyseisen teknologian soveltuvuus tehtävään. Näistä teknologioista oli tarjolla laajoja, erilaisista näkökulmista tehtyjä teknisiä suorituskykytestejä. Testit osoittivat, että valitut teknologiat edustivat vahvasti luokkiensa kärkipäätä. Konfiguraatioiden tallennus ja päivitys ei ole kuitenkaan järjestelmää kuormittava prosessi, joten tehokas toiminta suuren käyttöpaineen alla ei ollut tärkeimpiä mitta-arvoja teknologioiden valinnassa. Tämän vuoksi pääpaino oli luotettavuudessa ja helppokäyttöisyydessä. Monille teknologioille tehdyt testit oli toteutettu miljoonien kirjotettavien datapalasien avulla ja oli selvää, että kehitettävä järjestelmä ei joutuisi niin suuria määriä dataa käsittelemään. Kyseessä olevat teknologiat ovat hyvin moneen tarkoitukseen soveltuvia. Näin ollen testitulosten lukeminen auttaa hahmottamaan kunkin teknologian tehokkuuden. Mikäli tulevaisuudessa tarvitaan tämänkaltaisia teknologioita, on meillä jo kokemusta eri teknologioiden tehokkuudesta suuren käyttöpaineen alla.

2.1.1 Apache ZooKeeper ei ole tarpeeksi tehokas

Ensimmäinen teknologia, joka aluksi näytti sopivalta ja pätevältä, oli Apache ZooKeeper. Tätä teknologiaa käytetään melko laajalti suosituissa palveluissa kuten Yahoo ja Reddit. Apachen ZooKeeper olisi varmasti ollut toimiva vaihtoehto konfiguraatioiden tallennukseen. Suurimmiksi ongelmiksi nousivat kuitenkin sen tehovaatimukset. Apache ZooKeeper on myös kokonaisuutena hyvin laaja. Apache ZooKeeper toimii varmasti silloin, kun tarvitaan skaalautuva ja tehokas teknologia tallentamaan suuria määriä dataa. Kehitettävään järjestelmään tämän kokoluokan teknologia on kuitenkin aivan liian järeä. Java-pohjaisena on selvää, että tehovaatimukset ovat suuria verrattuna muilla kielillä kehitettyihin järjestelmiin. Apache ZooKeeperin käyttöönotto on myös melko monimutkainen prosessi verrattuna yksinkertaisempiin teknologioihin. Monimutkaisuus korostuu, mikäli asennuksesta ei ole aiempaa kokemusta. Toinen merkittävä asia ZooKeeperin hylkäämisessä oli yrityksen aikaisempi kokemus sen käytöstä. Aikaisemmissa projekteissa ZooKeeper oli noussut esille teknologioiden valintatutkimusten yhteydessä. Tässä vaiheessa se oli kuitenkin hyllytetty epävarmuuden, monimutkaisuuden ja sopivimpien teknologioiden löytymisen yhteydessä. Ei haluttu ottaa käyttöön jo kerran hylättyä teknologiaa vaan haluttiin löytää uudempi ja tehokkaampi ratkaisu, joka sopisi juuri tähän projektiin (Gurney 2017).

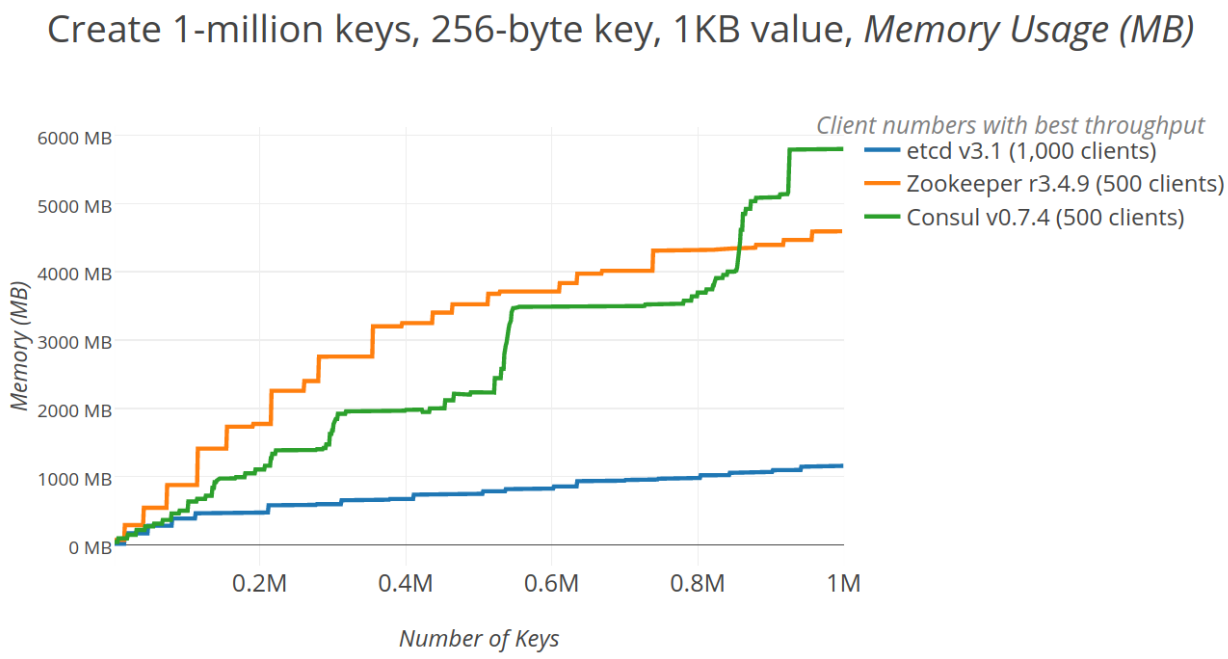
2.1.2 Consul tarjoaa ylimääräisiä toiminnallisuuksia

Apache ZooKeeperin tutkinnan jälkeen löytyi melko lupaavan oloinen Consul, joka on HashiCorpin kehittämä teknologia. Consulin avulla voidaan tallentaa avain-arvo-pareja. Consul tarjoaa myös service discoveryn. Service discovery on toiminnallisuutena eräänlainen palveluiden kauppapaikka. Käytännössä kyseinen toiminnallisuuden avulla palvelut markkinoivat itseään ja tarjoamiaan toimintoja sekä voivat löytää itselleen hyödynnettäviksi kelpaavia toisia palveluita. Tämä vähentäisi nykyisten konfiguraatioiden määrää, sillä melko suuri määrä konfiguraatioista on erinäisten palveluiden osoitteita (Lyaruu 2015). Consul alkoi näyttää lupaavalta, mutta tarkemman tutkimisien ja tarkastelun jälkeen selvisi, miten suuren määrän erilaisia palveluita se tarjoaa. Service discovery ja palveluiden health check ovat esimerkkejä toiminnallisuuksista, joita tässä projektissa ei tarvita. Ideana oli kuitenkin, että yksittäiset teknologiat tarjoavat rajatun määrän palveluita,

jotta lopullinen järjestelmä ei sisällä suuria määriä palveluita. Ylimääräiset palvelut toisivat mukanaan ei-haluttua kuormitusta. Varsinainen service discovery -osuus olisi myös ollut teknisesti melko haastavaa, jotta se olisi ollut järkevä toteuttaa. Consul siis päätettiin unohtaa toistaiseksi.

2.1.3 Etcd on tehokas ja yksinkertainen

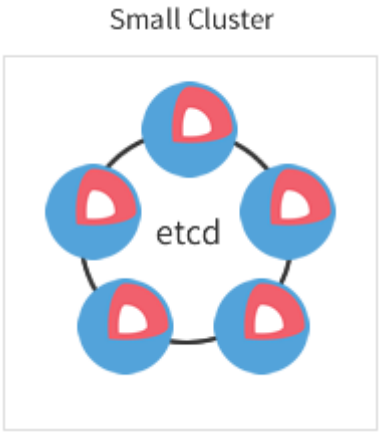
Lisätutkimusten jälkeen esille nousi Etcd, joka on CoreOS:n kehittämä avainarvoparien tallennukseen tarkoitettu järjestelmä. Etcd tarjoaa myös valmiiksi REST-rajapinnat, joiden avulla tiedon tallennus, muokkaus ja haku sujuvat helposti ja yksinkertaisesti. REST-rajapinnat ovat tietyn mallin mukaisia verkkorajapintoja, joihin voidaan lähettää eri tyyppisiä kutsuja sen mukaan, minkä palvelun halutaan käsittelevän kyseisen pyynnön. Pyynnöstä myös selviää, mitä lähetetyllä datalla tulisi tehdä. Etcd:n kehitys perustuu yksinkertaisuuteen ja tehokkuuteen. Tämä on erittäin toivottu ominaisuus. Erityisesti tehokkuudessa Etcd päihitti kilpailijoitaan (kuva 1).



KUVA 1. Etcd:n muistinkulutus (Exploring Performance of etcd 2016.)

Jälkeenpäin Etcd:n valinta osoittautui vielä paremmaksi, kun mukaan ilmestyi salasanojen kryptaukseen tarkoitettu Vault. Etcd:n käyttöönotto oli helppoa ja vaivatonta. Etcd on kirjoitettu käyttäen Go-ohjelmointikieltä, joka on tunnetusti kevyt ja tehokas pienten ohjelmien kirjoittamiseen tarkoitettu kieli. Näin ollen huomattavasti pienemmät tehokustannukset olivat ilmeiset esimerkiksi verrattuna Apache ZooKeeper -ohjelmistoon. Etcd on

myös erittäin helppo käyttää. Sen käyttöönotto on nopeaa ja asennukseen menee aikaa vain muutamia minuutteja. Dokumentaatio on kattava ja se tarjoaa kohta kohdalta ohjeet, joiden avulla voidaan pystyttää toimiva Etcd-instanssi. Etcd kykenee myös tarpeen vaatiessa klusterointiin (kuva 2). Klusteri tarkoittaa sitä, että useampi yksittäinen Etcd-instanssi kommunikoi keskenään ja jakaa tietoa. Jos yksi instanssi sammuu, voidaan klusteroinnin avulla saada edelleen dataa muista instansseista.



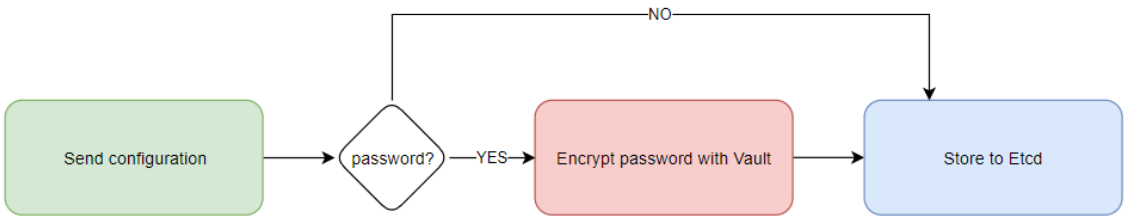
KUVA 2. Etcd-klusteri (CoreOS n.d.)

2.1.4 Vault käsittelee turvallisesti arkaluontoista tietoa

Osa konfiguraatioista on luonnollisesti hyvin arkaluontoista tietoa, jota ei pitäisi koskaan säilyttää ilman kryptausta. Kryptaus tarkoittaa annetun selkokielisen arvon muuttamista siten, että ihminen ei sitä enää pysty lukemaan. Kryptaukseen käytetään yleisesti monimutkaisia matemaattisia algoritmeja. Yleensä kryptatun tiedon palauttamiseen tarvitaan myös pitkä, satunnaisesti luotu avain. Nykyisessä konfiguraatiossa kryptaus ei ole aivan välttämätöntä, sillä tieto sijaitsee suoraan palveluiden palvelimilla, joihin ei ole ulkopuolista pääsyä. Mikäli konfiguraatiot pidettäisiin keskistetyssä palvelussa, tulisi kaikki salasanat kryptata mahdollisen tietovuodon takia. Tähän tehtävään löytyi vastaus jo aikaisemmin, vaikka sitä ei huomattu. Kun tutkittiin Consulin sopivuutta varsinaiseen konfiguraatioiden tallennukseen, esille nousi saman yrityksen (HashiCorp) tuottama Vault.

Vault toimii hyvin samalla tavalla kuin Etcd, erona se, että Vault kryptaa tiedon, joka sille syötetään. Vaultin sopivuus kryptaukseen vahvistui, kun selvisi, että se pystyy suoraan integroitumaan Etcd:n kanssa (kuva 3), jolloin salasanat käytännössä kulkevat Vaultin kautta ja tallentuvat kryptattuna Etcd:n varastoon. Tällöin salasanoille ei tarvita erillistä säilytyspaikkaa, vaan ne voidaan varastoida muiden konfiguraatioiden kanssa yhdessä

paikassa. Vault tarjoaa mahdollisuuden helposti muuttaa kryptatun tiedon tallennuspaikkaa, joten haluttaessa voidaan salasanat säilöä korkeamman turvallisuuden omaavalla palvelimella.



KUVA 3. Salattavien konfiguraatioiden toimintalogiikka

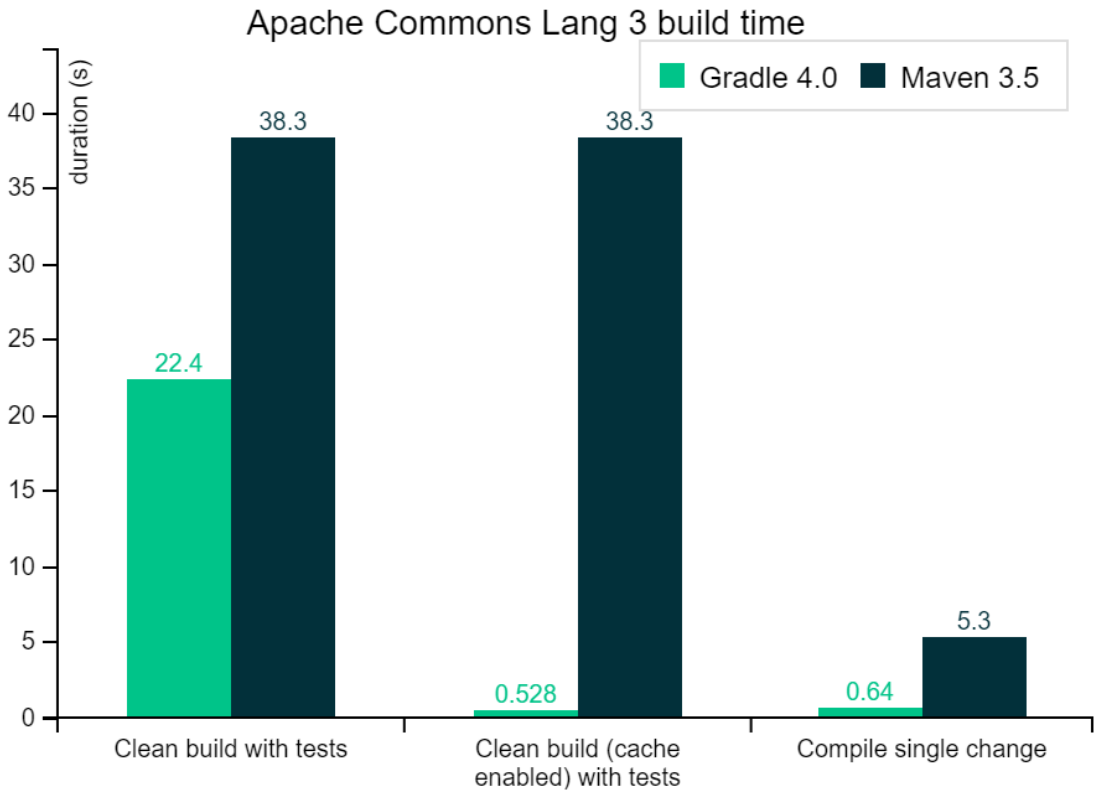
2.1.5 Gradle on Mavenia nopeampi build tool

Java-ohjelmistojen kehityksessä oleellista on työkalu, jolla koodi voidaan paketoida valmiiksi sovellukseksi. Tämä build tool on keskeisessä osassa, kun halutaan saada kirjoitettu koodi paketoitua jar-paketiksi. Tuotettua pakettia voidaan hyödyntää tuotantoasennuksissa. Toimeksiantajan kehittämässä ohjelmistossa tätä tehtävää hoitaa Maven. Maven on toimiva ratkaisu ja laajalti käytössä. Mavenin avulla voidaan hallita koko ohjelmistokomponenttia. Se tarjoaa käyttöön työkaluja, joiden avulla esimerkiksi paketointi ja versiointi on helppoa hallita. Monet näistä ominaisuuksista ovat erittäin toimivia. Mavenin pääasiallinen tehtävä kehityksessä on koodin paketointi toimivaksi moduuliksi. Tämä tapahtuu kirjoittamalla pom.xml tiedosto projektiin (kuva 4). Pom.xml-tiedosto sisältää ohjeet, joiden mukaan Maven hoitaa komponentin paketoinnin. Siinä kerrotaan muun muassa tuotettavan komponentin nimi, versio ja riippuvuudet (Introduction to the POM). Riippuvuuksilla tarkoitetaan muita moduuleja, joita tämä kyseinen moduuli tarvitsee, jotta se voi toimia oikein. Moduulit voivat olla kehittäjän itse valmistamia tai ne voivat olla kolmannen osapuolen valmistamia.

```xml
<project>
  <!-- Project info -->
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>
  <packaging>pom</packaging>

  <!-- Project modules -->
  <modules>
    <module>../my-module</module>
  </modules>

  <!-- Project dependencies -->
  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-artifact</artifactId>
      <version>${mavenVersion}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-project</artifactId>
      <version>${mavenVersion}</version>
    </dependency>
  </dependencies>
</project>
```

KUVA 4. Esimerkki pom.xml:n sisällöstä.

Gradle vastaa toimintaperiaatteeltaan Mavenia. Se sisältää myös tarvittavia työkaluja, joiden avulla koodi saadaan paketoitua. Merkittävä ero kehittäjälle tulee Gradlen vaatimassa ohjeistuksessa. Siinä missä Maven käyttää hyödykseen pom.xml tiedostoa, käyttääkin Gradle puolestaan omanlaisellaan syntaksilla kirjoitettua build.gradle-tiedostoa. Liittestä 1 voi nähdä toteutetun järjestelmän build.gradle-tiedoston. Vaikka tämä tiedosto näyttää suppeammalta, sisältää se kuitenkin kaikki samat tiedot, joita myös Mavenin käyttämä pom.xml sisältää. Merkittävin ero tulee kuitenkin itse paketointiajassa. Gradle on huomattavasti nopeampi kuin Maven (kuva 5). Gradle on jokaisessa tilanteessa vähintään kaksi kertaa Mavenia nopeampia ja monessa tapauksessa lähes sata kertaa nopeampi (Gradle vs Maven Comparison). Äärimmäinen nopeus oli syy, miksi Gradle päätettiin ottaa käyttöön.



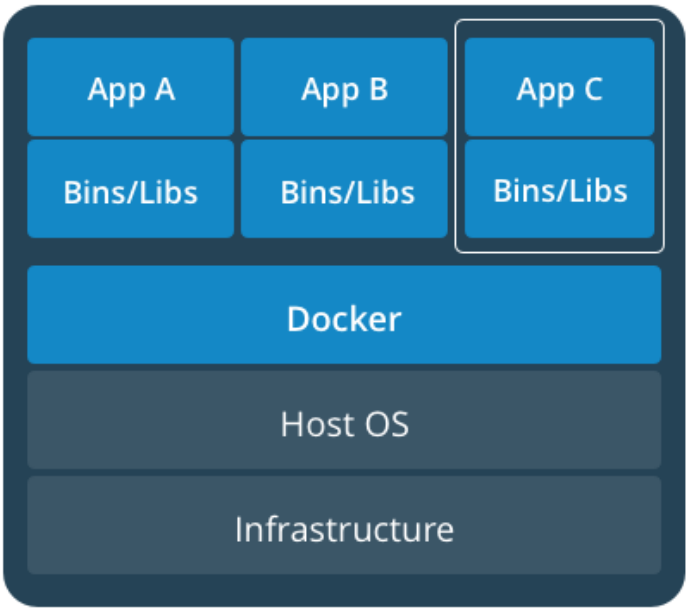
KUVA 5. Gradle vs Maven paketointiajat (Gradle vs Maven Comparison 2017)

2.2 Ympäristön teknologiat

Pian oli selvää, että mikäli halutaan hyödyllinen ja tuotantokelpoinen järjestelmä, pitää sen olla helposti pystytettävä ympäristö, joka voidaan tarvittaessa asentaa ilman suurempia ongelmia. Näiden teknologioiden osalta valinta oli melko yksinkertainen. Haluttiin noudattaa yrityksessä käytössä olevia periaatteita, miten ja millä teknologioilla ympäristöt toteutetaan. Näin ollen omat kokemukset kyseisitä teknologioista olivat jo tasolla, jolla pystyin käyttämään niitä tehokkaasti. Opinnäytetyöprosessin aikana tietämys näistä teknologioista tietysti kasvoi syvemmän tarkastelun ja tutkimisen ansiosta. Osa teknologioiden valinnoista oli siis jo perusteltuja, sillä ne tulisivat tulevaisuudessa auttamaan työtehtävissäni. Tämän vuoksi monen teknologian kohdalla ei ollut edes tarpeellista tutkia mahdollisia kilpailijoita.

2.2.1 Docker ja docker-compose avuksi hallitsemaan virtuaalisten palvelinten kokonaisuutta

Docker on ollut yrityksen käytössä jo pidemmän aikaa. Se on todettu toimivaksi ja tehokkaaksi tavaksi, kun pitää saada aikaan ympäristöjä, jotka voidaan tarpeen mukaan päivittää tai asentaa uusille palvelimille. Docker luo Docker-kontin, joka on eräänlainen virtuaalipalvelin. Docker-kontissa on erillinen käyttöjärjestelmä sekä omat sovelluksensa (kuva 6). Kontit myös luovat oman virtuaalisen verkon, jonka avulla useampi kontti voi olla yhteydessä toisiinsa. Docker-kontille luodaan oma kuvauksensa, Dockerfile. Tämä tiedosto sisältää kaiken tarvittavan. Se kertoo käyttöjärjestelmän, asennettavat ohjelmistot ja mitä kontin halutaan tekevän, kun se käynnistetään (esimerkiksi suorittaa tietyn prosessin). Tätä projektia varten näitä kontteja luotiin useita. Yrityksellä on ollut periaatteena, että yhdessä kontissa ajetaan vain yksi prosessi. Näin eri komponenttien päivitys ja virhetilanteet eivät vaikuta muiden prosessien kulkuun. On nopeampaa päivittää tai korjata vain yksi prosessi, kuin että koko järjestelmä jouduttaisiin käynnistämään uudestaan.



KUVA 6. Docker konttien rakenne (Docker 2017)

Docker-compose ei ole kovin monimutkainen teknologia. Se on käytännössä luotu hallitsemaan koko järjestelmää. Aikaisemmin mainitsin, että tässä projektissa tuotettu järjestelmä sisältää useita kontteja. Docker-composella voidaan helpommin ylläpitää näitä kontteja ja esimerkiksi ensimmäisessä asennuksessa yhdellä komennolla voidaan pystyttää ja linkittää kaikki järjestelmän kontit. Ja koska voidaan haluta hajauttaa järjestelmä useammille palvelimille, voidaan helposti pystyttää vain ne osat, jotka halutaan. Voidaan sanoa, että siinä missä Dockerfile oli yksittäisen kontin kuvaus, on docker-compose koko järjestelmän kaikkien konttien kuvaus.

2.2.2 Spring Boot luo helposti käynnistettävän palvelun

Työkokemuksen ja koulutuksen vuoksi oli selvää, että varsinainen logiikka toiminnanohjaukselle kirjoitetaan käyttäen Java ohjelmointikieltä. Oletuksena Java-prosessin käynnistys on kuitenkin haastava, varsinkin jos kyseessä on isompi järjestelmän osa. Tähän ongelmaan on kehitetty useita ratkaisuja, joista suuren suosion on saavuttanut Spring Boot. Spring Boot on luotu, jotta Springiä hyödyntävät Java-prosessit ovat mahdollisimman helppoja käynnistää. Spring on Java-ohjelmia varten luotu ohjelmistokehys. Avainasemassa Spring toimii riippuvuuksien automaattisessa täyttämisessä. Riippuvuuksien automaattisella täytöllä (dependency injection) tarkoitetaan sitä, että tarvittavat oliot luodaan automaattisesti niin, että niitä ei tarvitse luoda koodissa. Spring Bootilla tuotetaan yksi Jar-paketti, joka sisältää kaiken, mitä prosessi toimiakseen tarvitsee. Tämä paketti voidaan perinteisesti käynnistää java -jar -komennolla.

2.2.3 Nginx on erittäin joustava välityspalvelinratkaisu

Mikäli järjestelmää käytetään keskitetysti, se tarkoittaa, että palvelun olisi parasta olla yhden osoitteen takana. Nginx tarjoaa tähän tehokkaan ja yksinkertaisen ratkaisun. Nginx toimii käytännössä välityspalvelimena (proxy) kaikille järjestelmän eri osille. Se voidaan konfiguroida ohjaamaan liikenne haluttuun osoitteeseen järjestelmän sisällä (kuva 7). Lisäksi Nginx tarjoaa yksinkertaisen ratkaisun pääsynhallintaan. Otimme käyttöön basic access authentication -varmennuksen, jonka avulla ilman tunnuksia palvelimelle ja sitä kautta järjestelmään ei ole pääsyä. Nginx voidaan myös helposti laittaa tuottamaan lokia, jonka avulla voidaan seurata palvelimen liikennettä.

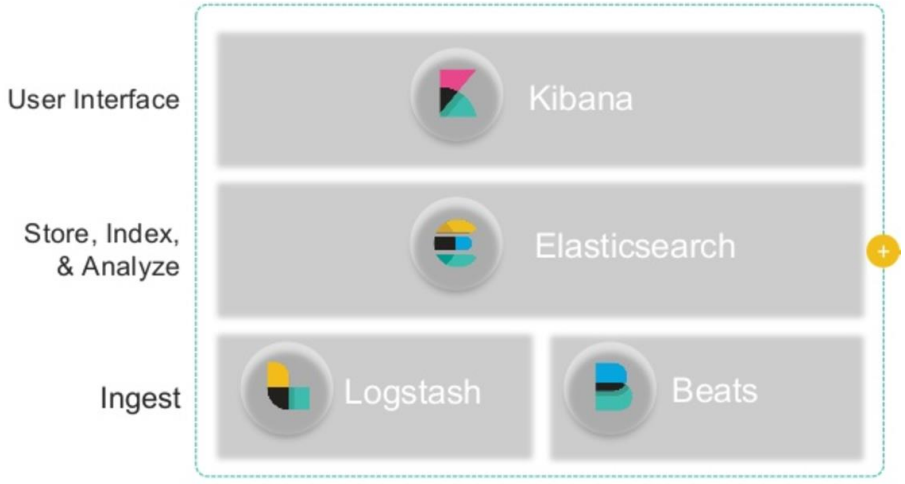
```
server {
    listen       80;
    server_name  localhost;

    access_log  /var/log/configuration-management/host.access.log  main;

    location / {
        proxy_pass http://config:8080;
    }

    location /webgui {
        proxy_pass http://webgui:3000;
    }
}
```

KUVA 7. Nginx konfiguraatio

2.3 Lokien keräys ja hallinta Elastic-stackilla

Lokien kerääminen järjestelmästä on erityisen tärkeää. Käytössä oleva Etcd ei suoraan pidä yllä tietoa siitä, mitä ja mistä tietoja on haettu. Haluttiin mahdollisuus seurata, mitkä palvelimet ja palvelut ovat noutaneet konfiguraatioita. Näin pystytään selvittämään, onko tietyllä palvelimella viimeisemmät, ajanmukaiset konfiguraatiotiedot.

Koska järjestelmän olisi hyvä noudattaa samoja käytäntöjä kuin muidenkin yrityksen järjestelmien, valittiin lokien keruuseen ja hallintaan Elastic-stack (kuva 8). Elastic-stack koostuu komponenteista, jotka keräävät, varastoivat, prosessoivat ja näyttävät niille annettuja lokitietoja (The Open Source Elastic Stack). Seuraavaksi käydään lyhyesti läpi, mistä teknologioista Elastic-stack koostuu. Syvempää läpikäyntiä ei ole tarkoituksenmukaista tehdä, sillä Elastic-stack on teknologiana erittäin monipuolinen ja joustava.



KUVA 8. Elastic-stackin rakenne (Wallez 2016)

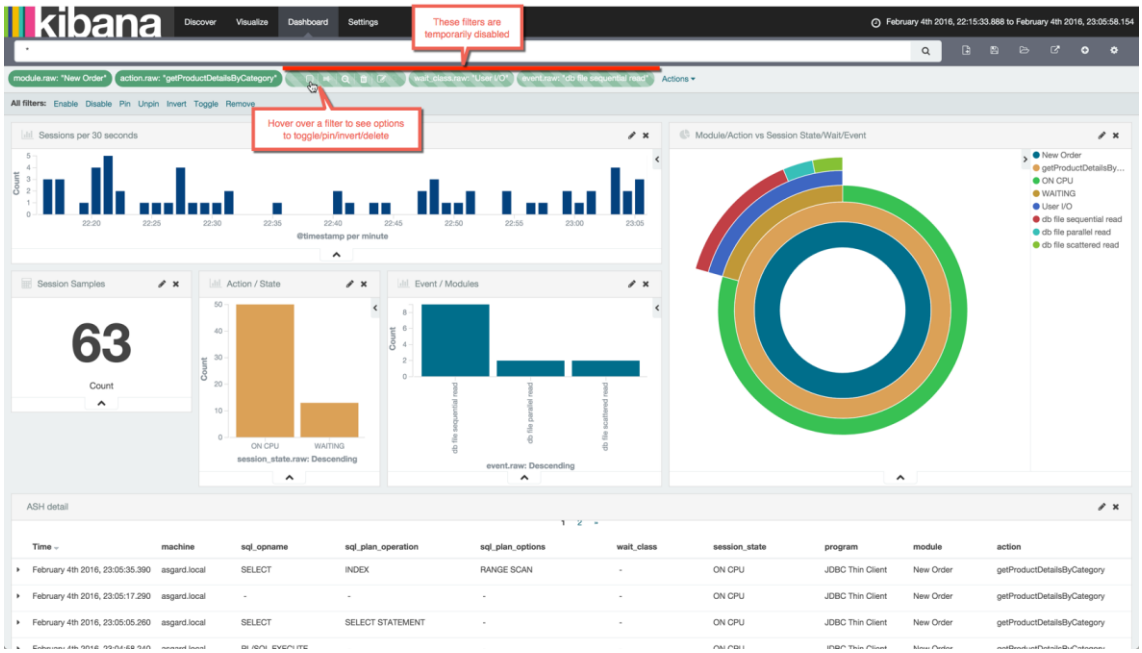
2.3.1 Elasticsearch on tehokas indeksointi- ja hakujärjestelmä

Elasticsearch on tehokas indeksointi-, haku- ja analysointiohjelmisto. Elasticsearch on ollut yrityksen käytössä erinäisissä tarkoituksissa. Elasticsearchin suurin hyöty on, että se perustuu indeksointiin, joka puolestaan nopeuttaa dataan kohdistettuja hakuja huomattavasti. Indeksointia voidaan verrata kirjoissa käytettäviin sisällysluetteloihin. Sen sijaan, että jotakin asiaa etsitään koko kirjasta, voidaankin etsiä haluttu kohta ensin sisällysluettelosta. Tämä tekee hakemisesta huomattavasti nopeampaa, sillä koko datan läpikäyntiä ei tarvita (Gormley & Tong 2015).

Tässä projektissa Elasticsearchia käytetään tallentamaan ja analysoimaan tuotettuja lokeja. Näin saadaan hyödyllistä tietoa esimerkiksi siitä, milloin tietty palvelin on viimeksi päivittänyt omat konfiguraationsa.

2.3.2 Kibana näyttää monipuolisesti Elasticsearchin dataa

Kibana on Elasticsearchiin liitettävä käyttöliittymä. Oletuksena Elasticsearch ei sisällä graafisia käyttöliittymiä, vaan se vastaa pelkästään datan indeksoinnista ja hakujen suorittamisesta. Raaka data ei ole ihmiselle hyödyllistä sellaisenaan, vaan se on tarkoitettu ohjelmien käyttöön. Datasta voidaan saada kuitenkin hyvin paljon hyötyä kuten tässä tapauksessa, jossa dataa käytetään selvittämään oleellisia asioita konfiguraatioiden käytöstä. Tätä varten on kehitetty Kibana, joka itsessään jo sisältää suuren määrän graafisia komponentteja datan näyttämiseen. Kibana on kehitetty varta vasten Elasticsearchin rinnalle näyttämään dataa ihmiselle sopivammassa muodossa. Kibana suurin vahvuus on sen helppo, mutta monipuolinen muokattavuus. Oletuksena Kibana ei ole muuta kuin Elasticsearchin indekseissä olevan datan listaus. Siihen voidaan tallentaa itse tehtyjä hakuja helposti opittavan syntaksin mukaan. Voidaan esimerkiksi hakea vain tulokset, joilla on tietty tilatieto, kuten poistettu tai erääntynyt. Näitä hakuja voidaan puolestaan hyödyntää graafisissa elementeissä. Kibana tarjoaa käyttäjälle yleisiä diagrammeja, kuten viiva-, pylväs- ja ympyrädiagrammin, johon voidaan liittää ennalta tehtyjä hakuja (kuva 9). Lopuksi hyödyntäen tehtyjä hakuja ja graafisia elementtejä voidaan rakentaa helposti dashboardeja. Dashboardit ovat käytännössä kokoelmia erilaisia diagrammeja ja hakutuloksia. Näitä dashboardeja voidaan sitten luoda erilaisia tarkoituksia varten. Verkkokauppasovelluksesta voitaisiin esimerkiksi näyttää tilausten eri tietoja tai varaston tilannetta tuotteista.



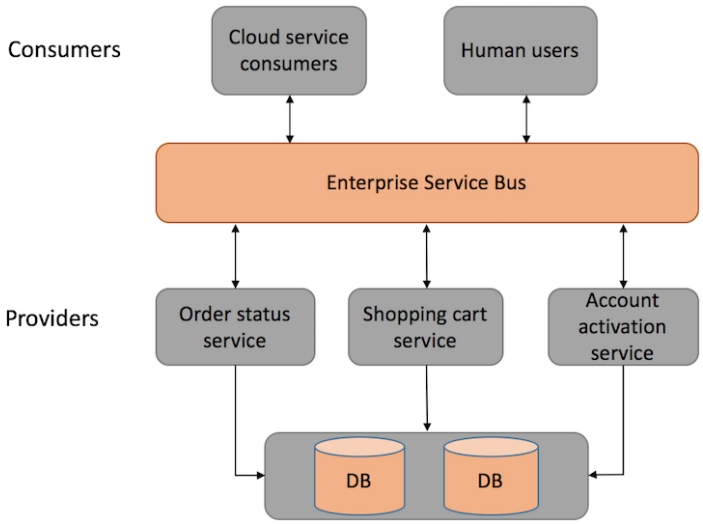
KUVA 9. Esimerkki Kibanan dashboardista. (Elasticsearch 2017)

2.3.3 Logstash välittää helposti palvelimen lokeja

Logstash on Elastic-stackin osa, joka ottaa vastaan dataa. Logstash prosessoi datan annetuilla filttereillä ja välittää sen edelleen Elasticsearchin indeksiin. Logstashin filtterit ovat omalla Grok-syntaksilla, joten sen opettelu saattaa viedä hieman aikaa. Grok-syntaksi muistuttaa kuitenkin läheisesti JSON-syntaksia, joten jos JSON on käyttäjälle tuttua, helpottaa se suuresti Logstashin käyttöä. Tietysti varsinkin yleisimpiin lokimuotoihin on saatavilla valmiita Grok-filttereitä verkosta. Logstashin avulla on mahdollista, että eri lähteistä tuleva data muokataan eri tavoilla ja lähetetään loppujen lopuksi oikean muotoisena oikeaan Elasticsearchin indeksiin. Näin esimerkiksi Nginx välityspalvelimen ja varsinaisen konfiguraatiohallintajärjestelmän lokit voidaan helposti erotella toisistaan. Logstash ei tarjoa minkäänlaista käyttöliittymää. Se on tehty mahdollisimman kevyeksi ja hyvä niin: tarkoituksena on, että Logstashin ainoa tehtävä on prosessoida raaka data Elasticsearchin indekseihin vietäväksi.

3 PALVELUKESKEINEN ARKKITEHTUURI

Opinnäytetyössä tuloksena syntyvä järjestelmä on kehitetty tukemaan hajautetun järjestelmän konfiguraatiohallintaa. Toimeksiantajan järjestelmä on kehitetty käyttäen palvelukeskeistä arkkitehtuuria (Service Oriented Architecture, SOA). Palvelukeskeisessä arkkitehtuurissa perusperiaatteena on jakaa suurempi järjestelmä pienempiin palveluihin. Sen sijaan, että luodaan yksi suuri järjestelmä, luodaankin pienempiä palveluita, jotka voivat tarjota omia toimintojaan muille palveluille. Näin syntyy palveluiden verkosto, joka muodostaa halutun järjestelmän. Palvelukeskeiseen arkkitehtuuriin kuuluu tiedon ja datan omistajuus. Esimerkiksi tilauspalvelu omistaa asiakkaiden tilausten tiedot ja voi tarjota niitä muille palveluille hyödynnettäväksi tarjoamiensa rajapintojen kautta (kuva 10). Koska järjestelmä koostuu itsenäisistä palveluista, on mahdollista, että palvelut käyttävät eri teknologioita toteutuksessa. Palvelut voivat sitten kommunikoida esimerkiksi REST-rajapintojen kautta. Palvelukeskeinen arkkitehtuuri ei ole sama kuin mikropalveluarkkitehtuuri. Huomattavimpina eroina ovat tietokannat sekä palveluiden laajuus. Palvelukeskeisessä arkkitehtuurissa palvelut käyttävät samoja tietokantoja, kun taas mikropalveluissa yleensä jokaisella palvelulla on oma tietokantansa. Myös palveluiden kokoluokissa on merkittäviä eroja. Mikropalvelut ovat huomattavasti pienempiä ja tekevät vain muutamia toimintoja, kun taas palvelukeskeisen arkkitehtuurin palvelut ovat kooltaan huomattavasti suurempia ja laajempia toiminnoiltaan. (Miri 2017)



KUVA 10. Palvelukeskeisen arkkitehtuurin rakenne. (Miri 2017)

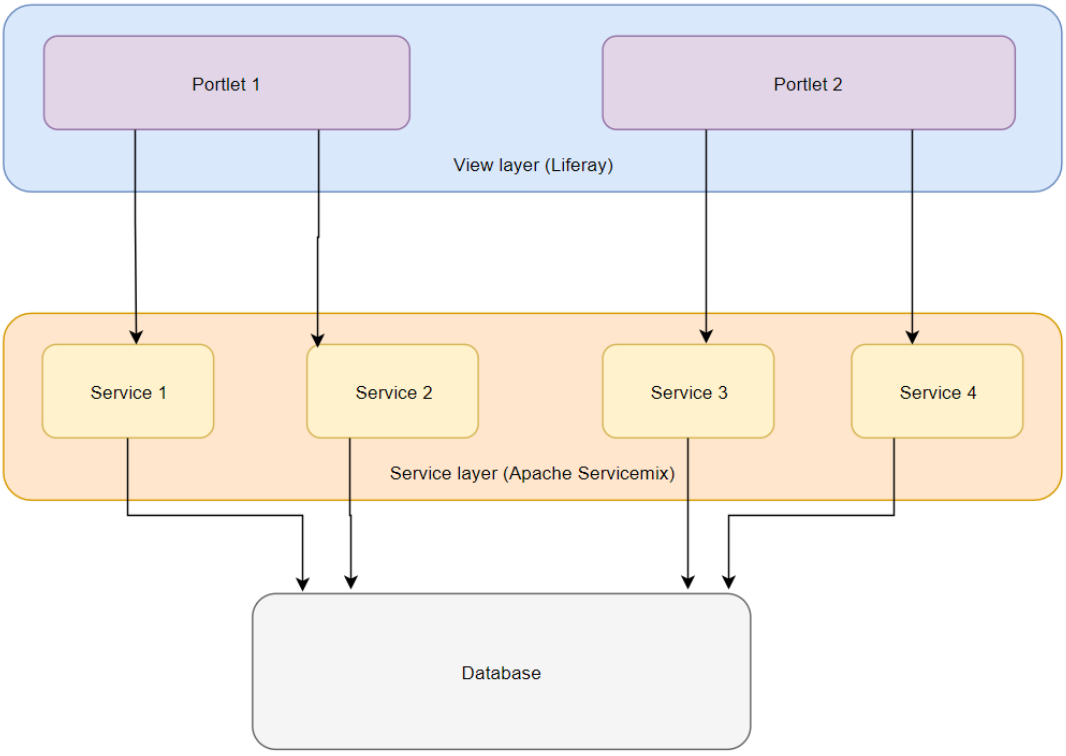
3.1 Järjestelmä, johon konfiguraatiohallinta liitetään

Tämän opinnäytetyön tarkoituksena oli luoda hajautetun järjestelmän konfiguraatiota helpottava ohjelmistokomponentti. Vaikkakin itse ohjelmistokomponentti oli tarkoitus toteuttaa hyvin geneeriseksi, tuli sen silti ensisijaisesti palvella toimeksiantajan kehittämää ja ylläpitämää järjestelmäkokonaisuutta. Kyseisen järjestelmäkokonaisuuden arkkitehtuuri tukeutuu hyvin vahvasti palvelukeskeiseen arkkitehtuuriin (kuva 11). Tässä opinnäytetyöraportissa käytetään esimerkkinä verkkokauppajärjestelmää.

Järjestelmän keskeinen osa on Apache Servicemix. Nimensä mukaisesti Servicemix on ohjelmisto, jonka on tarkoitus koota erillisiä palveluita yhteen. Tämä ajattelumalli tukee loistavasti palvelukeskeistä arkkitehtuuria. Servicemix sisältää palveluita, jotka tarjoavat omia palveluitaan muiden käytettäväksi. Servicemixin sisällä olevat palvelut voivat myös hyödyntää toisiaan, luoden näin toimivan järjestelmäkokonaisuuden. Erilaisia palveluita on näin ollen helpompi päivittää ja kehittää. Mikäli esimerkiksi tilauspalveluun tulee suuria muutoksia, voidaan sitä jatkokehittää itsenäisesti tai se voidaan jopa hankkia toisen kehittäjän kautta. Tämä mahdollistaa erillisten palveluiden kehittämisen eri aikaan ja koska palvelut ovat itsenäisiä komponentteja, ei tarvitse odottaa toisen, erillisen kehitystyön valmistumista. Servicemix muodostaa näin ollen palvelukerroksen järjestelmäkokonaisuudelle. Moniin järjestelmiin, kuten myös tähän, kuuluu oleellisena osana tietokanta, jossa voidaan säilyttää dataa. Servicemixin palvelut ovat edelleen yhteydessä tietokantaan.

Järjestelmän toinen merkittävä osa on Liferay-verkkoportaali. Liferay on hyvin yksinkertainen, mutta monipuoliset mahdollisuudet tarjoava verkkoalusta. Liferay tarjoaa jo valmiiksi yksinkertaiset työkalut, mikäli kehittäjä haluaa saada nopeasti aikaan yksinkertaisen verkkopalvelun, kuten blogi-, wiki- tai uutissivuston. Toimeksiantajan järjestelmissä näitä valmistyökaluja käytetään vähän ja pääpaino on kehittää omia portletteja. Portletit ovat ikään kuin pieniä sovelluksia, jotka tarjoavat käyttäjälle mahdollisuuden kommunikoida järjestelmän kanssa ja hyödyntää sen tarjoamia palveluita. Portletit puolestaan ovat yhteydessä aikaisemmin mainittuun Servicemixiin ja sen palveluihin. Näin ollen pystytään rakentamaan ketju, jolloin käyttäjä pääsee käsiksi tietokantojen datavarantoihin ja voi hyödyntää tätä palvelukokonaisuuksien tarjoamaa dataa ja palveluita.

Verkkokauppasovelluksessa Liferay saattaa sisältää esimerkiksi ostoskorisovelluksen, jonka avulla käyttäjä voi nähdä, lisätä ja muokata oman ostoskorinsa sisältöä. Ostoskorisovellus on puolestaan yhteydessä Servicemixin ostoskoripalveluun, joka edelleen tallentaa ja tarjoaa käyttäjälle hänen oman ostoskorinsa sisällön. Ostoskoripalvelu voi myös suorittaa muita ostoskoriin liittyviä toiminnallisuuksia, kuten laskea yhteishinnan sekä käyttäjän halutessa välittää tiedot edelleen tilauspalvelulle.



KUVA 11. Toimeksiantajan järjestelmän kuvaus

3.2 Konfiguraatio nykyisessä järjestelmässä

Yrityksen kehittämä ja ylläpitämä järjestelmä on monien muiden järjestelmien tapaan vahvasti konfiguroitavissa. Suurelta osin palveluita voidaan siis ohjata ilman varsinaista koodin muutosta. Esimerkkinä toimii aikaisemmin mainittu verkkokauppasovellus. Koska samaa järjestelmää voidaan markkinoida useammalla eri yritykselle, tulee sen olla helposti muokattavissa. Yritys A voi haluta, että varauksista lähtee vahvistuksena aina sähköposti automaattisesti asiakkaalle. Yritys B voikin haluta, että varausvahvistusta ei lähetetä. Nykyisessä toimeksiantajan järjestelmässä tämä on hoidettu konfiguraatiotiedostojen avulla. Erilaisiin tarkoituksiin on luotu normaaleja tekstitiedostoja, joissa jokaisella rivillä on yksi käytettävä konfiguraatioarvo, esimerkiksi sähköpostin lähettämisestä.

Kuten aikaisemmin on mainittu, myös eri palveluiden palveluosoitteet on kerrottu näissä tiedostoissa. Järjestelmä lukee käynnistyessään näitä konfiguraatioarvoja (kuva 12).

```
1  # Database basic
2  database.url=localhost:5200
3  database.user=username
4  database.password=pass123
5
6  # Cart service
7  cart.db.name=shopping_cart
8  cart.default.currency=EUR
9  cart.save.session=true
10
11 # Order service
12 order.confirmation.default=EMAIL
```

KUVA 12. Kuvitteellinen kuvaus konfiguraatiotiedoston sisällöstä

Konfiguraatioarvot voitaisiin siis lukea käytännössä suoraan tiedostoista. Tähän tarkoitukseen kuitenkin on käytössä Spring ja sen tarjoamat työkalut. Tarkoitus on, että palvelun käynnistyessä Spring luo automaattisesti palvelun tarjoamia olioita ja asettaa niille arvoja konfiguraatioiden perusteella. Nämä konfiguraatiot voidaan lukea useasta eri paikasta ja tässä tapauksessa ne luetaan konfiguraatiotiedostosta. Tämä tarkoittaa sitä, että konfiguraatiot haetaan vain palvelun käynnistyessä. Mikäli palvelun konfiguraatioihin täytyy tehdä muutoksia, tehdään ne ensin konfiguraatiotiedostoon ja tämän jälkeen käynnistetään palvelu uudestaan, jolloin uudet konfiguraatiot tulevat voimaan. Luonnollisesti tällöin palvelua ei voida käyttää ja yleisin tapa näissä tilanteissa on pitää palvelussa käyttökatko. Käyttökatko tarkoittaa, että käyttäjille ilmoitetaan palvelun olevan poissa käytössä tietyn ajan. Tämän käyttökatkon aikana on tarkoituksena muuttaa tarvittavat konfiguraatiot, päivittää palveluita uusiin versoihin ja käynnistää palvelut uudestaan.

3.3 Ongelmakohdat nykyisessä toimintamallissa

Ongelmakohtia nykyisessä järjestelmässä selvitettiin yhteistyössä järjestelmän asiantuntijoiden kanssa. Nykyinen toimintamalli konfiguraatiotiedostojen kanssa on selvästi tehoton. Palvelimelle siirtyminen, tiedoston paikantaminen ja sen muuttaminen vievät paljon aikaa. Lisäksi työ tehdään komentorivillä. Komentorivin työkaluilla tiedostoa muokatessa virheiden tapahtuminen on todennäköisempää, kuin jos työ tehtäisiin graafisilla työkaluilla. Tästä päädyttiin ratkaisuun, että graafiset työkalut helpottaisivat toimintaa.

Tarvitaan helppokäyttöinen hallintanäkymä, jotta konfiguraatioita ei tarvitse hallinnoida komentoriviltä.

Ilman erillistä hallintatyökalua ei voida myöskään varmistaa, sisältääkö tiedosto kaikki tarpeelliset konfiguraatiot. Ei voida myöskään taata, että se ei sisällä ylimääräisiä konfiguraatioita esimerkiksi edellisten versioiden vuoksi. Näiden seuraamista varten tarvitaan työkalut. On selvää, että on tehotonta, mikäli konfiguraatioiden tarpeellisuutta ei voida taata. Myös jälkikäteen lisättävien puuttuvien konfiguraatioiden työmäärä on tarpeeton, mikäli voitaisiin alusta asti todeta puuttuva konfiguraatio.

Koska konfiguroitavia ympäristöjä on useita, aiheuttaa se myös ylimääräistä työtaakkaa. Hajautetuissa järjestelmissä ei ole taattua, että kaikki palvelimet ovat yhteydessä toisiinsa. Yleensä tietyillä palvelimilla pääsyt ovat hyvin rajattuja. Tämän vuoksi jokainen palvelin sisältää itsessään omat konfiguraationsa. Jos järjestelmään kuuluu useita palvelimia, tarkoittaa niiden konfiguraatioiden hallinta palvelimelta toiselle liikkumista. Tarvitaan keskitetty paikka, josta jokainen palvelin voi itse noutaa konfiguraatiot. Näin riittää, että muutos tehdään yhteen järjestelmään sen sijaan, että jouduttaisiin tekemään sama muutos usealle palvelimelle.

4 TOTEUTETTU JÄRJESTELMÄ

Järjestelmän toteutus alkoi varsinaisen ongelman kartoittamisella. En itse ollut työtehtävissäni ollut mukana järjestelmien ylläpidossa tai asennuksissa. Näin ollen en varsinaisesti ollut kohdannut ongelmaa, johon haettiin ratkaisua. Pidimme useita palavereja kartoittaen ongelman ydintä. Toimeksiantaja osoitti erittäin paljon panostusta ongelman ratkaisuun, sillä toimiva ratkaisu helpottaisi erittäin paljon varsinaisia ylläpitotehtäviä. Nopeimmillaan oltaisiin tietysti voida luoda yksinkertainen järjestelmä, joka ei tee muuta kuin ajaa asiansa. Haluttiin kuitenkin luoda paremmin suunniteltu järjestelmä, jossa on potentiaalia jatkokehitykseen ja mahdollisuuksien mukaan jopa tuotteistamiseen asti tulevaisuudessa. Näin ollen varsinainen suunnitteluvaihe ja sen onnistuminen olivat erittäin kriittistä. Haluttiin, että järjestelmän kehitys on menossa oikeaan suuntaan heti alkumetreiltä asti. Muutamien suunnittelupalaverien jälkeen ongelma alkoi selvitä itselleni paremmin ja sain konkreettisemman kuvan siitä, minkälainen ongelma tuotettavalla järjestelmällä halutaan ratkaista.

Konkreettisen kuvan saaminen ongelmasta antoi varsinaisen lisäsysäyksen itse järjestelmän kehittämiselle. Alettiin pohtimaan toimivan järjestelmän logiikkaa. Lähtökohtana oli pitää järjestelmä hyvin minimaalisena. Haluttiin myös, että se ei olisi riippuvainen muista järjestelmistä, vaan että sitä voitaisiin hyödyntää helposti tarvituissa paikoissa. Tässä vaiheessa myös geneerisyys astui vahvasti kuvaan. Konfiguraatioiden tarve on kuitenkin melko yleinen asia ja haluttiin, että tuotettu järjestelmä toimii itsenäisenä kokonaisuutena ja on näin ollen helposti käytettävissä myös muissa projekteissa.

4.1 Konfiguraatioiden tallennus Etcd:llä

Etcd:n tarjoama yksinkertainen ratkaisumalli tallentamiselle on omiaan yksinkertaisten konfiguraatioiden tallentamiseen. Perinteiseen tietokantaan verrattuna se tuo kuitenkin mukanaan omia haasteitaan. Koska tallennus tapahtuu valmiiksi määritetyllä tavalla, ei tallennettaviin konfiguraatioihin voida lisätä itse määritettyä lisätietoa. Toimeksiantajan järjestelmissä lähes kaikella datalla on lisätietona oleellisia asioita. Tällaisia asioita ovat esimerkiksi tieto siitä kuka on lisännyt tai muokannut dataa ja milloin. Etcd ei suoraan tukenut tällaista mahdollisuutta. Ongelmaan ei kuitenkaan etsitty ratkaisua, sillä tämän hetken tilanteessa konfiguraation muutoksista ei myöskään jää jälkeä.

Oleellinen asia tallennuksen osalta on, että data on helposti hahmotettavissa. Vaikka nykyisellään kaikki konfiguraatiot ovat tekstinä tiedostoissa, on tiedostot kuitenkin nimetty kuvaamaan konfiguraatioiden kohteita. Etcd tarjoaa tähän valmiin ratkaisun. Etcd:llä on mahdollista luoda kansioita ja alikansioita. Kansiot ja alikansiot erotetaan yksinkertaisesti konfiguraation avaimesta. Aivan alussa on juuritaso, jossa ei ole minkäänlaista etuliitettä. Kun kansioita ja alikansioita luodaan, ilmestyy avaimeen tiedot siitä, missä kansiossa kyseinen konfiguraatio sijaitsee (kuva 13).

```
1  /Root
2  /Root/Folder
3  /Root/Folder/Configuration1
4  /Root/Folder/Configuration2
5  /Root/Folder/Subfolder
6  /Root/Folder/Subfolder/Configuration3
```

KUVA 13. Esimerkki tuotetusta konfiguraatioiden hierarkiarakenteesta.

Tätä mahdollisuutta päätettiin käyttää luomaan hierarkia, joka mahdollistaa konfiguraatioiden tunnistamisen. Näin voidaan erotella esimerkiksi eri asiakkaiden konfiguraatiot ja eri käyttötarkoituksen konfiguraatiot, vaikka ne sijaitsevatkin samassa Etcd-instanssissa.

4.2 Toiminnan ohjaus Spring Boot -applikaatiolla

Etcd itsessään tarjoaa valmiit REST-rajapinnat. Suoraan käytettynä niiden kanssa joutuisi kuitenkin keksimään monimutkaisia toimintatapoja, koska ne eivät tarjoa suoraan toiminnallisuuksia, joita tuotettavassa järjestelmässä tulisi olla. Tähän tarkoitukseen luotiin palvelu, joka toimii Spring Boot -applikaationa. Tämä järjestelmän osa on vastuussa kaikesta logiikasta, jota tuotettavan järjestelmän tulee noudattaa. Erityisen hyvä puoli tässä on se, että taustalla toimivaa tallennusratkaisua voidaan tarvittaessa muuttaa. Lisäksi voidaan kehittää uusia toiminnallisuuksia. Nämä uudet toiminnallisuudet puolestaan voivat käyttää jotain aivan erilaista taustaratkaisua. Nyt ne ovat kuitenkin helposti yhden konfiguraatioihin keskittyneen palvelun tarjoamina ja voidaan luottaa, että samat REST-rajapinnat tarjoavat halutun tiedon.

Spring Boot -applikaatioon on helppo rakentaa REST-rajapintoja. Nämä ovat toimeksiantajalla nykyisellään standardi tapa luoda palveluiden tarjoamia rajapintoja. Spring Boot tarjoaa helpoilla Java-annotaatioilla mahdollisuuden luoda REST-rajapinnan eri osia,

jotka tarjoavat järjestelmän toiminnallisuuksia muiden ohjelmistojen käyttöön (kuva 14). Tätä järjestelmää varten kehitettiin esimerkiksi rajapintoja konfiguraatioiden luomiseen, muokkaamiseen ja hakemiseen.

```
@RestController
public class ConfigurationManagementRest extends ApplicationRest {
    private Logger log = LoggerFactory.getLogger(ConfigurationManagementRest.class);
    @Autowired
    ConfigurationManagementService configurationManagementService;

    /* CONFIG */
    @CrossOrigin
    @RequestMapping(value = "/config", method = GET)
    public String value(@RequestParam String key,
                        @RequestParam(required = false, defaultValue = "true") boolean history) {
        log.info("GET /config [key: {}, history: {}]", key, history);
        EtcdResponse response = configurationManagementService.getValue(key, history);
        return success(response);
    }

    @CrossOrigin
    @RequestMapping(value = "/config", method = POST, consumes = "application/json")
    public String store(@RequestBody EtcdNodeDTO configuration) {
        log.info("POST /store [key: {}, value: {}]", configuration.getKey(), configuration.getValue());
        EtcdResponse response = configurationManagementService.storeConfiguration(configuration);
        return success(response);
    }
```

KUVA 14. REST-rajapinta Spring Boot -applikaatiossa.

Spring Boot tuottaa valmiin helposti ajettavan Java-applikaation. Tämän vuoksi tämän komponentin käynnistäminen on erittäin helppoa. Palvelin tarvitsee vain Java-asennuksen, jonka avulla voidaan käynnistää tuotettu jar-paketti.

4.3 Nginx-välityspalvelin

Toimivan palvelinkokonaisuuden edellytyksenä on luoda pääsy hallitseva ja seuraava välityspalvelin, joka toimii porttina kaikkiin muihin palvelimen toiminnallisuuksiin. Nginx konfiguraatio on yksinkertaista. Järjestelmän eri osioihin tehdään ohjaukset. Tämän ansiosta yhden osoitteen takana voi olla useampia järjestelmiä riippuen siitä, mihin polkuun pyyntö on lähetetty. Näin ollen esimerkiksi www.testiosoite.com/hallinta ja www.testiosoite.fi/haku voivat viedä eri palvelimen osioihin. Näille voidaan myös määrittää omat autentikointitiedostot. Nginx:n mukana tuleva basic access authentication -varmennus voidaan asettaa hakemaan vaaditut tunnukset sen mukaan, minne käyttäjä haluaa päästä.

Tuotettuun järjestelmään rakennettiin tämän Nginx-välityspalvelimen avulla pääsy kolmeen eri osioon. Ensimmäiseksi luotiin reitti konfiguraatiohallintapalvelun rajapintoihin. Nämä rajapinnat eivät ole käyttäjille avoinna, vaan ne on tarkoitettu yksinomaan eri ohjelmien ja palveluiden käytettäviksi. Toinen ohjattava reitti on konfiguraatiohallinan

verkkokäyttöliittymä. Tänne on asetettu pääsy vain pääkäyttäjille, sillä sieltä voidaan hallita konfiguraatioiden arvoja. Viimeinen ohjattava reitti vie Kibanaan. Kibanaan on annettu pääsy pääkäyttäjille ja ylläpitäjille.

Nginx konfiguroitiin kirjoittamaan lokia kaikista sen kautta kulkevista pyynnöistä. Tätä lokia voidaan lähettää eteenpäin ja sen avulla voidaan seurata tarkasti, mitä palvelussa on tehty ja kuka sen on tehnyt.

4.4 Lokien valvominen Elastic-stackissa

Lokeista saadaan paljon hyödyllistä tietoa. Kuten aikaisemmin mainittu, Etcd ei suoraan tallenna historiatietoja tai tietoa siitä kuka käyttää palvelua. Nginx asetettiin kirjoittamaan näistä lokia. Tämä loki välitetään suoraan Elasticsearchin indeksiin. Kaikki tapahtumat tallennetaan sellaisenaan. Tämä mahdollistaa lokien tarkemman analysoinnin. Kun lokeja tallennetaan, ei ole kannattavaa karsia mitään pois. Ajatus siitä, että ”Tätä ei todennäköisesti tarvita” voi johtaa huonoon tilanteeseen. Pois heitettyä dataa ei enää saada takaisin, jos sitä halutaan hyödyntää.

Kibana voi näyttää tallennetun datan käyttäjälle ihmiselle sopivalla tavalla. Tuotetun järjestelmän osalta on oleellista tietää, mikä palvelu on hakenut konfiguraatioita. Tässä oleellista tietoa on myös se, koska konfiguraatio on haettu. Nykyisen konfiguraatiotiedoston heikkous piilee hyvin vahvasti myös siinä, että suoraan ei nähdä koska ja mitä on tiedostosta muokattu. Kibanan avulla voidaan nopeasti ja helposti tarkistaa, onko asiakkaan ympäristössä konfiguraatiot ajan tasalla.

5 POHDINTA

Tavoitteena oli kehittää nykyistä toimintamallia konfiguraatioiden osalta toimeksiantajan järjestelmissä. Nykyinen malli oli todettu tehottomaksi, mutta itselläni ei ollut täsmällistä kuvaa, minkä vuoksi. En ollut työtehtävissä joutunut käsittelemään asiakasasennuksien konfiguraatioita. Näiden ongelmien selvittäminen oli mielenkiintoista ja asiantuntijat olivat avuliaasti mukana. Tästä sai sen kuvan, että opinnäytetyölläni on toimeksiantajalle ja sen työntekijöille oikeaa konkreettista hyötyä. Tarkoituksena oli luoda toimiva ohjelmistokomponentti, joka auttaa tulevaisuudessa konfiguraatiohallintaa. Alusta asti sain tukea ja resursseja. Nämä mahdollistivat syvemmän paneutumisen aiheeseen. Ongelmaan oli olemassa olevia ratkaisuja, mutta ne eivät olleet täysin sitä, mitä haluttiin. Materiaali itse aiheesta ei ollut kovin kattavaa, joten omaa harkintaa ja pohtimista yhdessä asiantuntijoiden kanssa joutui harrastamaan paljon.

Konfiguraatiot varsinkin suurissa järjestelmissä ovat osoittautuneet tärkeiksi. Niiden avulla voidaan helposti ja nopeasti muuttaa ohjelmiston toimintaa ilman koodiin tehtäviä muutoksia, jotka puolestaan vaatisivat järjestelmän uuden asentamisen. Mikäli konfiguraatioiden muokkaaminen tuottaa liian suuren määrän työtä, on toimintamalliin ja niiden hallintaan hyvä kiinnittää huomiota. Tehokas konfiguraatiohallinta voi säästää lukuisia työtunteja ja näin ollen tehostaa kehittäjän työskentelyä. Konfiguraatioiden tärkeys korostuu, kun järjestelmä on hajautettu usealle palvelimelle. Tällöin keskitetty järjestelmä konfiguraatioiden hallintaan ja jakamiseen on erinomainen vaihtoehto.

Opinnäytetyöprosessi eteni ajallaan. Koska työlläni oli suuri merkitys toimeksiantajalle, sain mahdollisuuden tehdä suurimman osan varsinaisesta ohjelmistokomponentista työajalla. Sain käyttööni tarpeelliset resurssit, jotta koko työn tekeminen olisi edes mahdollista. Prosessin aikana taitoni käytettyjen teknologioiden käytössä syvenivät. Monien jo aiemmin tuntemieni teknologioiden käytöstä minulla oli yleistä kokemusta. Opinnäytetyön aikana jouduin kuitenkin syventymään niihin ja tämä varmasti tulee auttamaan tulevaisuudessa. Opin myös paljon itse ympäristöjen asennuksista. Jouduin opinnäytetyön aikana paljon tekemisiin palvelinten asentamisien ja palvelinympäristöjen suunnitellun kanssa.

Onnistumista osoittivat myös tulevaisuuden suunnitelmat kehitetyn ohjelmistokomponentin osalta. Lisätoiminnallisuuksia sekä käyttötarkoituksia on listattu suuret määrät.

Toteutettu järjestelmä osoittautui geneerisemmäksi kuin alun perin oli odotettu. Vaikka järjestelmä kehitettiin yhden tietyn järjestelmäkokonaisuuden konfiguraatiohallintaan, osoitti se kykyä toimia monenlaisissa muissa ympäristöissä. Yksi merkittävä käyttökohde löytyi yrityksen sisältä. Toteutettua järjestelmää pyritään soveltamaan yrityksen työntekijöiden tietojen säilömiseen, jolloin esimerkiksi tarvittavat työympäristön konfiguraatiot voitaisiin säilöä toteutettuun järjestelmään. Järjestelmää voitaisiin käyttää myös säilyttämään ja jakamaan erinäisten peruspalvelinten asennuksiin vaadittavia tietoja.

LÄHTEET

Apache. N.d. Introduction to the POM.
https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

CoreOs. N.d. CoreOS Container Linux cluster architectures
https://coreos.com/os/docs/latest/cluster-architectures.html

Docker Inc. 2017. What is a Container
www.docker.com

Elasticsearch BV. 2017. The Open Source Elastic Stack.
https://www.elastic.co/products

Gormley, C & Tong, Z. 2015. Elasticsearch: The Definitive Guide. O'Reilly Media

Gradle Inc. 2017. Gradle vs Maven Comparison.
https://www.gradle.org/maven-vs-gradle/

Gurney, J. Liiketoimintajohtaja. 2017. Haastateltu 2017. Haastattelija Timonen, J. Tampere

Lee, G. 2017. Exploring Performance of etcd, Zookeeper and Consul Consistent Key-value Datastores
https://www.coreos.com/blog/performance-of-etcd.html

Lyaruu, F. 2015. Service Discovery in OSGi: Beyond the JVM using Docker and Consul. https://www.slideshare.net/FrankLyaruu/javacro

Miri, I. 2017. Microservices vs SOA.
https://www.dzone.com/articles/microservices-vs-soa-2

Wallez, S. 2016. Kibana + timelion: time series with the elastic stack
https://www.slideshare.net/swallez/kibana-timelion-time-series-with-the-elastic-stack

LIITTEET

Liite 1. Tuotetun ohjelmistkomponentin build.gradle -tiedosto.

1 (2)

```
group 'com.eduix.configuration.management'
version '1.0-SNAPSHOT'

buildscript {
    repositories {
        mavenCentral()
        mavenLocal()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:1.5.2.RELEASE")
    }
}

apply plugin: 'java'
apply plugin: 'maven'
apply plugin: 'eclipse'
apply plugin: 'idea'
apply plugin: 'org.springframework.boot'

jar {
    baseName = 'configuration-management-impl'
    version = '1.0-SNAPSHOT'
}

repositories {
    mavenCentral()
    mavenLocal()
}

sourceCompatibility = 1.8
targetCompatibility = 1.8
```

```
dependencies {
    compile("net.logstash.log4j:jsonevent-layout:1.0")
    compile("org.springframework.boot:spring-boot-starter-web") {
        exclude group: 'org.springframework.boot', module: 'spring-boot-starter-logging'
    }
    compile group: 'org.springframework.boot', name: 'spring-boot-starter-log4j', version: '1.3.8.RELEASE'
    compile group: 'org.slf4j', name: 'jcl-over-slf4j', version: '1.7.25'
    compile project(":configuration-management-api")
    compile group: 'org.apache.httpcomponents', name: 'httpclient', version: '4.5.2'
    compile group: 'com.google.code.gson', name: 'gson', version: '2.8.0'
    testCompile(' org.springframework.boot: spring - boot - starter - test')
}
```